Apache Flink
Next-gen data analysis

Kostas Tzoumas
ktzoumas@apache.org
@kostas_tzoumas
What is Flink

• Project undergoing incubation in the Apache Software Foundation

• Originating from the Stratosphere research project started at TU Berlin in 2009

• [http://flink.incubator.apache.org](http://flink.incubator.apache.org)

• 58 contributors (doubled in ~ 4 months)

• Has a cool squirrel for a logo 🐿️
This talk

• Data processing engines in Hadoop

• Flink from a user perspective

• Tour of Flink internals

• Closing
DATA PROCESSING IN THE HADOOP ECOSYSTEM
Open source data infrastructure – an era of choice

Applications
- Hive
- Cascading
- Mahout
- Pig
- ... 

Data processing engines
- MapReduce
- Flink
- Spark
- Storm
- Tez

App and resource management
- Yarn
- Mesos

Storage, streams
- HDFS
- HBase
- Kafka
- ...
Engine paradigms & systems

MapReduce (OSDI'04)

Dryad, Nephele (EuroSys'07)

PACTs (SOCC’10, VLDB’12)

RDDs (HotCloud’10, NSDI’12)

Apache Hadoop 1

Apache Tez

Apache Flink (incubating)

Apache Spark
**Dryad**

- Small recoverable tasks
- Sequential code inside map & reduce functions
- Extends map/reduce model to DAG model
- Backtracking-based recovery

**Spark**

- Functional implementation of Dryad recovery (RDDs)
- Restrict to coarse-grained transformations
- Direct execution of API

- Embed query processing runtime in DAG engine
- Extend DAG model to cyclic graphs
- Incremental construction of graphs
# Engine comparison

<table>
<thead>
<tr>
<th>API</th>
<th>MapReduce on k/v pairs</th>
<th>k/v pair Readers/Writers</th>
<th>Transformations on k/v pair collections</th>
<th>Iterative transformations on collections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paradigm</td>
<td>MapReduce</td>
<td>DAG</td>
<td>RDD</td>
<td>Cyclic dataflows</td>
</tr>
<tr>
<td>Optimization</td>
<td>none</td>
<td>none</td>
<td>Optimization of SQL queries</td>
<td>Optimization in all APIs</td>
</tr>
<tr>
<td>Execution</td>
<td>Batch sorting</td>
<td>Batch sorting and partitioning</td>
<td>Batch with memory pinning</td>
<td>Stream with out-of-core algorithms</td>
</tr>
</tbody>
</table>
USING FLINK
Data sets and operators

Program

Parallel Execution
Rich operator and functionality set

Map, Reduce, Join, CoGroup, Union, Iterate, Delta Iterate, Filter, FlatMap, GroupReduce, Project, Aggregate, Distinct, Vertex-Update, Accumulators
**WordCount in Java**

```java
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

DataSet<String> text = readTextFile (input);

DataSet<Tuple2<String, Integer>> counts = text
    .map (l -> l.split("\W+"))
    .flatMap ((String[] tokens, Collector<Tuple2<String, Integer>> out) -> {
        Arrays.stream(tokens)
            .filter(t -> t.length() > 0)
            .forEach(t -> out.collect(new Tuple2<>(t, 1)));
    })
    .groupBy(0)
    .sum(1);

env.execute("Word Count Example");
```
WordCount in Scala

```scala
val env = ExecutionEnvironment .getExecutionEnvironment

val input = env.readTextFile(textInput)

val counts = text 
  .flatMap { l => l.split("\W+") }
  .filter { t => t.nonEmpty }
  .map { t => (t, 1) }
  .groupBy(0)
  .sum(1)

env.execute()
```
Long operator pipelines

```
 DataSet<Tuple...> large = env.readCsv(...);
 DataSet<Tuple...> medium = env.readCsv(...);
 DataSet<Tuple...> small = env.readCsv(...);

 DataSet<Tuple...> joined1 = large
    .join(medium)
    .where(3).equals(1)
    .with(new JoinFunction() { ... });

 DataSet<Tuple...> joined2 = small
    .join(joined1)
    .where(0).equals(2)
    .with(new JoinFunction() { ... });

 DataSet<Tuple...> result = joined2
    .groupBy(3)
    .max(2);
```
Beyond Key/Value Pairs

```java
DataSet<Page> pages = ...;
DataSet<Impression> impressions = ...;

DataSet<Impression> aggregated =
    impressions
    .groupBy("url")
    .sum("count");

pages.join(impressions).where("url").equalTo("url")

// custom data types

class Impression {
    public String url;
    public long count;
}

class Page {
    public String url;
    public String topic;
}
```
Beyond Key/Value Pairs

Why not key/value pairs

• Programs are much more readable ;-) 

• Functions are self-contained, do not need to set key for successor)

• Much higher reusability of data types and functions
  – Within Flink programs, or from other programs
“Iterate” operator

- Built-in operator to support looping over data
- Applies step function to partial solution until convergence
- Step function can be arbitrary Flink program
- Convergence via fixed number of iterations or custom convergence criterion
Transitive closure in Java

```java
DataSet<Tuple2<Long, Long>> edges = ...

IterativeDataSet<Tuple2<Long, Long>> paths = edges.iterate(10);

DataSet<Tuple2<Long, Long>> nextPaths = paths
    .join(edges)
    .where(1).equalTo(0)
    .with((left, right) -> {
        return new Tuple2<Long, Long>(
            new Long(left.f0),
            new Long(right.f1));
    })
    .union(paths)
    .distinct();

DataSet<Tuple2<Long, Long>> transitiveClosure = paths.closeWith(nextPaths);
```
Transitive closure in Scala

```scala
val edges = ...

val paths = edges.iterate (10) {
  prevPaths: DataSet[(Long, Long)] =>
  prevPaths
    .join(edges)
    .where(1).equalTo(0) {
      (left, right) =>
        (left._1,right._2)
    }
    .union(prevPaths)
    .groupBy(0, 1)
    .reduce((l, r) => l)
}
```
"Delta Iterate" operator

- Compute next workset and changes to the partial solution until workset is empty
- Similar to semi-naïve evaluation in datalog
- Generalizes vertex-centric computing of Pregel and GraphLab
Using Spargel: The graph API

```java
ExecutionEnvironment env = getExecutionEnvironment();

DataSet<Long> vertexIds = env.readCsv(...);
DataSet<Tuple2<Long, Long>> edges = env.readCsv(...);

DataSet<Tuple2<Long, Long>> vertices = vertexIds
    .map(new IdAssigner());

DataSet<Tuple2<Long, Long>> result = vertices.runOperation(
    VertexCentricIteration.withPlainEdges(
        edges, new CCUpdater(), new CCMessager(), 100));

result.print();
env.execute("Connected Components");
```

Pregel/Giraph-style Graph Computation
Spargel: Implementation

Spargel is implemented in < 500 lines of code on top of delta iterations
Hadoop Compatibility

Flink supports out-of-the-box supports

- Hadoop data types (writables)
- Hadoop Input/Output Formats
- Hadoop functions and object model
FLINK INTERNALS
val paths = edges.iterate(maxIterations) {
  prevPaths: DataSet[(Long, Long)] =>
  val nextPaths = prevPaths
    .join(edges)
    .where(1).equalTo(0) {
      (left, right) => (left._1,right._2)
    }
    .union(prevPaths)
    .groupBy(0,1)
    .reduce((l,r) => l)
  nextPaths
}

Client

Optimization and translation to data flow

Distributed architecture

Task Manager

Scheduling, resource negotiation, ...

Job Manager

Data node

Task Manager

Memory heap

Data node

Task Manager

Memory heap

Data node
The growing Flink stack...

Scala API (batch)  Java API (streaming)  Apache MRQL

Python API

Graph API („Spargel“)

Java API (batch)

Common API

Flink Optimizer  Streams Builder

Java Collections  Apache Tez

Hybrid Batch/Streaming Runtime

Storage Streams  Files HDFS S3 JDBC Azure Kafka Rabbit MQ Redis ...

Local Execution  Cluster Manager  Native YARN EC2

Batch  Streaming
Program lifecycle

```scala
val source1 = ...
val source2 = ...
val maxed = source1 .map(v => (v._1, v._2, math.max(v._1, v._2))
val filtered = source2 .filter(v => (v._1 > 4))
val result = maxed .join(filtered) .where(0) .equalTo(0) .filter(_1 > 3) .groupBy(0) .reduceGroup { … }
```
Memory management

- Flink manages its own memory
- User data stored in serialize byte arrays
- In-memory caching and data processing happens in a dedicated memory fraction
- Never breaks the JVM heap
- Very efficient disk spilling and network transfers
**Little tuning or configuration required**

- Requires no memory thresholds to configure
  - Flink manages its own memory
- Requires no complicated network configs
  - Pipelining engine requires much less memory for data exchange
- Requires no serializers to be configured
  - Flink handles its own type extraction and data representation
- Programs can be adjusted to data automatically
  - Flink’s optimizer can choose execution strategies automatically
Understanding Programs

Visualize and understand the operations and the data movement of programs

Screenshot from Flink’s plan visualizer
Understanding Programs

Analyze after execution (times, stragglers, ...)

WebLogAnalysis Example

Scheduled: 10/4/2014 6:30:03 PM
Runtime: 1 sec 265 msecs
Status: FINISHED

Flow Layout  Stack Layout

9999  SCHEDULED  FINISHED

9

DataSource ([url_0,dolor ad amet enim laoreet nostrud veniam aliquip ex nonummy c

8

DataSource ([url_2,2003-12-17], [url_9,2008-11-11], [url_14,2003-11-5], [url_46,20k

7

DataSource ([url_30,url_0,43], [url_2,1.39], [url_2,31], [url_3,36], [url_3,36], [url_4,36], [url_5,u

6

CHAIN Filter (org.apache.flink.examples.java.relational.WebLogAnalysis$FilterDocByKeyV

5

CHAIN Filter (org.apache.flink.examples.java.relational.WebLogAnalysis$FilterVisitsByDate) -> Map (Project

4

Filter (org.apache.flink.examples.java.relational.WebLogAnalysis$FilterByRank)

3

Join(org.apache.flink.api.java.operators.

2

CoGroup (org.apache.flink.examples.j.

1

DataSink(Print to
Understanding Programs

Analyze after execution (times, stragglers, ...)

Tasks

CoGroup (org.apache.flink.examples.java.relational.WebLogAnalysis$AntiJoinVisits)

<table>
<thead>
<tr>
<th>localhost_3</th>
<th>deploying</th>
<th>running</th>
</tr>
</thead>
<tbody>
<tr>
<td>localhost_2</td>
<td>deploying</td>
<td>running</td>
</tr>
<tr>
<td>localhost_1</td>
<td>deploying</td>
<td>running</td>
</tr>
<tr>
<td>localhost_0</td>
<td>deploying</td>
<td>running</td>
</tr>
</tbody>
</table>

Filter (org.apache.flink.examples.java.relational.WebLogAnalysis$FilterByRank)

Join (org.apache.flink.api.java.operators..)

CoGroup (org.apache.flink.examples.j.)

DataSink (Print to)
Built-in vs driver-based iterations

Loop outside the system, in driver program
Iterative program looks like many independent jobs

Dataflows with feedback edges
System is iteration-aware, can optimize the job
Delta iterations

Cover typical use cases of Pregel-like systems with comparable performance in a generic platform and developer API.

Computations performed in each iteration for connected communities of a social graph

*Note: Runtime experiment uses larger graph*
Optimizing iterative programs

- Pushing work "out of the loop"
- Caching Loop-invariant Data
- Maintain state as index
WIP: Flink on Tez

- Flink has a modular design that makes it easy to add alternative frontends and backends
- Flink programs are completely unmodified
- System generates a Tez DAG instead of a Flink DAG

First prototype will be available in the next release
Closing
Upcoming features

• Robust and fast engine
  – Finer-grained fault tolerance, incremental plan rollout, optimization, and interactive shell clients

• Alternative backends: Flink on *
  – Tez backend

• Alternative frontends: * on Flink
  – ML and Graph functionality, Python support, logical (SQL-like) field addressing

• Flink Streaming
  – Combine stream and batch processing in programs
“Flink Stockholm week”

Wednesday
- Flink introduction and hackathon @ 9:00
- Flink talk at Spotify (SHUG) @ 18:00

Thursday
- Hackathons at KTH (streaming & graphs) @ 9:00